WEST Search History

Hide Items | Restore | Clear | Cancel

DATE: Thursday, February 19, 2004

Hide?	Set Name	Query	Hit Count
	DB=PGPB,USPT,USOC; PLUR=YES; OP=ADJ		
☐	L14	L13 and l3	21
☐	L13	L12 and l5	115
☐	L12	L4 and (identif$ near2 (node or intent$))	160
☐	L11	L5 and (identif$ near2 (node or intent$))	115
☐	L10	L8 and l3	21
☐	L9	L8 and syntax?independent	6
☐	L8	L5 and (identif$ near2 (node or intent))	115
☐	L7	L5 and l3	69
	DB=EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=ADJ		
☐	L6	L5	4
	DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=ADJ		
☐	L5	L4 and node and tree	326
☐	L4	program$ near (construct or intent)	1753
	DB=PGPB,USPT,USOC; PLUR=YES; OP=ADJ		
☐	L3	L2 or l1	6278
☐	L2	707/100-102.ccls.	4502
☐	L1	717/114-119,141-147,154-161.ccls.	1851

END OF SEARCH HISTORY

Hit List

Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

Search Results - Record(s) 1 through 4 of 4 returned.

☐ 1. Document ID: NN880556

L6: Entry 1 of 4 File: TDBD May 1, 1988

TDB-ACC-NO: NN880556

DISCLOSURE TITLE: Canonical Representation and Processing Algorithm for Recursive Logic Queries

PUBLICATION-DATA:
IBM Technical Disclosure Bulletin, May 1988, US

VOLUME NUMBER: 30
ISSUE NUMBER: 12
PAGE NUMBER: 56 - 64

DISCLOSURE TEXT:

- A technique is described which uses a canonical representation form, for recursive logic queries, and a processing algorithm that provides efficient processing through existing relational database optimizers in a loosely-coupled computer system. The canonical form deals with logic queries in function-free "Horn-clause" logic extended for negation. An extended disjuctive normal form (EDNF) representation of logic queries allows efficient processing in loosely coupled environments, but only covers the class of logic queries that are expressed in the function-free "Horn-clause" logic that include recursions involving the queried goal (Query goal). The concept, described herein, presents the canonical form of logic queries that covers most queries in the function-free "Horn-clause" logic extended for negation. The queries include arbitrary recursions, not necessarily involving the Query goal and may involve negations, which cannot normally be expressed in "Horn-clause" logic. Also, the concept provides an extension of relational database languages to include arbitrary recursions. The concept does not cover cases in which negation occurs in a recursion, since this type of query rarely occurs. In the processing of a general recursive query, there are units of information that must be obtained before processing other units; thus there is a partial order in processing these units. Each unit is called a "forest" and is represented in the EDNF. It consists of a set of noncyclic and cyclic trees. A forest must be instantiated before other forests are evaluated. A temporary table is assigned to a forest to ***** SEE ORIGINAL DOCUMENT ***** maintain evaluation results. However, exceptions are provided to handle subqueries, so that a forest can be evaluated, as a subquery, without having to create a temporary table. A forest is formed whenever there is a recursion, which does not involve the Query goal or when there is a negation. In addition, since the result of the Query goal is the answer to be obtained, the Query goal is always designated as a forest. Typically, there is more than one goal in a cycle, and any one of them can represent the forest. A specific goal that represents the forest is defined as the "forest head". Since a forest must be fully evaluated before other forests can be evaluated, there is an inherent partial order in processing the forests. An order between two forests is established if there is a directed path between the two. A forest graph represents this partial ordering, and the evaluation or the

query must be done according to this partial order. The concept is illustrated using an example utilizing the following rules: r1: t(X,Y) <-m(X,Y) & *(X,Y) r2: n(X,Y) <-1(X,Z) & q(Z,Y) r3: t(X,Y) <-e(X,Z) & p(Z,Y) r4: p(Z,Y) <-e(Z,L) & p(L,Y) r5: p(Z,Y) <-a(Z,L) & t(L,Y) ***** SEE ORIGINAL DOCUMENT ***** Suppose query ?t(X,5) is issued, then the program graph will appear as shown in Fig. 1. In this case, three forests are needed: one for the Query goal, one for the negated goal, and one for the cycle at P. The EDNF of each forest is shown in Fig. 2. The subscript F denotes that a node is a forest rather than a base table. The algorithm which obtains the EDNF stops expansion if it encounters a forest. This is performed because a forest represents a fully instantiated node. An improvement can be obtained by bypassing forests that are not part of cycles. Also, a forest created due to negation can be bypassed if it is not traversed through the negation edge. For example, the forest p1, as shown in Fig. 2, was obtained by bypassing the forest t1 . Therefore, the partial order among forests t1, n1 and p1 is obtained, as shown in Fig. 3. If a forest does not form a strict partial order and form a cycle among them, it cannot be evaluated independently; instead, all the forests in the cycle must be evaluated iteratively until a fixed point is reached. The set of forests connected by cycles is defined as a "metaforest", usually formed when there are multiple cycles in the program graph that are interconnected with one another. Also, a single forest, not involved in a cycle, is a metaforest. Therefore, a graph of metaforests is truly a tree and represents the partial order of evaluation among metaforests. As in the forest graph, an edge is found between two metaforests, if there is a directional path between the two with the direction of the edge being the same as that of the path. ***** SEE ORIGINAL DOCUMENT ***** An example of a program graph involving multiple interconnected cycles is shown in Fig. 4. It is drawn in a simplified form, where goals t, p and u are chosen as forest heads. The corresponding EDNF of each forest, t1, p1 and u1, is shown in Fig. 5. The forest graph for the program of Fig. 3 is shown in Fig. 6, and the metaforest graph is shown in Fig. 7. Choosing forest heads in a given program graph for optimization is performed so as to reduce the number of forests. An example of choosing forest heads is shown in Fig. 8, where a program graph involves multiple cycles with one node covering all cycles. A simple way of choosing forest heads is to pick t, p and u (one goal node for each cycle) as the forest heads. However, this will create three forest heads, and forests p1F and u1F will form a metaforest for which a fixed point operation is performed. If s is chosen instead of p and u, since s is included in both cycles, one forest head will be sufficient. The resulting forest, the EDNFs for forests t1 and s1, is shown in Fig. 9. ***** SEE ORIGINAL DOCUMENT ***** From this example, the number of forests is minimized by choosing goals that are included in as many cycles as possible, therefore, a set of nodes covers multiple interconnected cycles if each cycle contains at least one node in the set. The objective is to find the smallest set of nodes that covers the interconnected cycles. The canonical form of a query is a graph of forests, each forest being represented in the EDNF. The following two algorithms are used: a) for constructing the canonical form of a query, and b) for processing the logic query, based on the canonical form: Algorithm for constructing the canonical form Algorithm Canonical(Logic Query) 1. Construct_program_graph 2. Detect_forest_heads 3. For each forest head (FH) do Call EDNF(FH, Forest, Identity_mapping, False,Identity_mapping, FH) 4. Construct the forest graph 5. Construct the metaforest graph Detect_forest_heads: Find the set of forest heads for program graph as follows: 1. For each cluster of cycles interconnected together, find a minimal set of nodes that covers all cycles; designate each node in the set as a forest head. 2. For each negated edge in the program graph, designate the node at the tail side as the forest head. Mark it noncyclic if it is not included in the cycle. 3. Designate Query goal as a forest. Mark it as noncyclic if it is not included in a cycle. Construct_forest_graph: Construct the graph of forests as follows: 1. Create a forest node for each forest head. 2. If there is a directed path between any two forest heads in the program graph, then create a directed edge between them, with the direction of the edge being the same as that of the directed path. Construct_metaforest_graph: Construct the graph of forests as follows: 1. Create a metaforest node for each forest in the forest graph that is not in any cycle. 2. If there is a cluster of interconnected cycles, create a metaforest node that includes all the forests connected by these interconnected cycles. 3. If there is an edge between any two forests in different metaforests, then create a directed edge between them, with the direction of the edge

being the same as that of the edge between the two forests. The algorithm EDNF for obtaining an EDNF form of a forest in a general recursive query is described as follows, using the notations: Edgemap: Mapping function described on an edge. The function maps the variable name on the head side of the edge to the variable name on the tail side of the edge. Accmapping: Accumulative mapping function that maps a variable name to another variable name for a unique path between a node and the node where this mapping chin begins. - Accmapping is obtained as a composition of Edgemaps along this path. For example, if variable X in Node 1 was mapped to Y in Node 2, the variable Y in Node 2 to Y in Node 3 and the variable Y in Node 3 to Z in Node 4, then Accmapping(X) = Z at Node 4. - Forest: EDNF for a forest head in consideration. - Origin__forest__head: The head for which the EDNF is being obtained. - This becomes the root of the EDNF. - Node: A node in the program graph Negation: Flag indicating that the edge being traversed is negated. - Algorithm EDNF(Node, Forest, Edgemap, Negation, Accmapping, Origin__forest__head) For every variable (X) in Node newmapping(X) := Edgemap (Accmapping) If node is a goal node Then If Node is a leaf node or (a cyclic forest head but not origin__forest__head) or (a forest head and Negation = true) Then Replace__variable__names Tree =Construct__tree(Node,Origin__forest__head) If Negation = true Then tag Node as a negation node Forest : =Tree Else if Node is Origin__forest__head and this is not the first visit to Origin__forest__head Then Replace__variable__names Tree =Construct__tree(Node,Origin__forest__head) Tag the tree as a cyclic tree Forest :=Tree Else /*nonleaf node or (noncyclic forest head and Negation=false) or Origin__forest__head at the first visit*/ Forest :=been | For each child (C) of Node and the corresponding edge

☐ 2. Document ID: US 6097888 A

L6: Entry 2 of 4 File: DWPI Aug 1, 2000

DERWENT-ACC-NO: 2000-654992
DERWENT-WEEK: 200366

TITLE: Executable code generation method for computer program in computer system, involves transforming node of intentional program tree into implementation of new high level computational construct

INVENTOR: SIMONYI, C

PRIORITY-DATA: 1995US-0431049 (April 28, 1995), 1993US-0145689 (October 29, 1993)

PATENT-FAMILY:

PUB-NO	PUB-DATE	LANGUAGE	PAGES	MAIN-IPC
US 6097888 A	August 1, 2000		059	G06F009/45

INT-CL (IPC): G06 F 9/45

ABSTRACTED-PUB-NO: US 6097888A
BASIC-ABSTRACT:

NOVELTY - The intentional program tree is created by directly manipulating the program tree by a programmer. The creation adds a node to program tree representing new high level computational construct. The node is then transformed into implementation of new high level computational construct. The implementation is represented by one or more nodes of low level computational construct.

DETAILED DESCRIPTION - The intentional program tree having nodes is received. The nodes represent a high level computational construct of computer program. The node is transformed into implementation of high level computational construct. The implementation is represented by one or more nodes of low level computational construct. An executable code that implements the low level computational construct is generated. INDEPENDENT CLAIMS are also included for the following:

(a) computer system for generating computer program;

(b) executable code generation program

USE - For supporting extensible computational constructs used in creating computer programs, generally written in high level programming language like Pascal and C.

ADVANTAGE - Since new high level computational constructs are defined, each new high level computational construct represents a programmers' intent and is used when creating an intentional program tree.

DESCRIPTION OF DRAWING(S) - The figure shows the block diagram of the components of the IP system.

☐ 3. Document ID: US 6078746 A

L6: Entry 3 of 4 File: DWPI Jun 20, 2000

DERWENT-ACC-NO: 2000-585972
DERWENT-WEEK: 200366

TITLE: Operand determining method for operator, during computer program generation, involves identifying a selected subtree as operand and identifying an operand according to preset rules if subtree is not selected

INVENTOR: SIMONYI, C

PRIORITY-DATA: 1995US-0431049 (April 28, 1995), 1993US-0145689 (October 29, 1993), 1997US-0884202 (June 27, 1997)

PATENT-FAMILY:

PUB-NO	PUB-DATE	LANGUAGE	PAGES	MAIN-IPC
US 6078746 A	June 20, 2000		055	G06F009/445

INT-CL (IPC): G06 F 9/445

ABSTRACTED-PUB-NO: US 6078746A

BASIC-ABSTRACT:

NOVELTY - Sequence of tokens associated with program tree is received. When an indication of next token to be appended to the sequence is received, a subtree is selected and identified as an operand of operator represented by the next token. If a subtree is not currently selected, an operand is identified according to preset rules. A node indicating the operator and the identified operand is then added.

DETAILED DESCRIPTION - The predefined rules of operator precedence used for identifying an operand if a subtree is not currently selected, can be overridden by selection of a subtree INDEPENDENT CLAIMS are also included for following: the program product; editor in computer system.

USE - For determining an operand for an operator during data entry of the nodes of an intentional program tree, in a computer system, when generating a computer program.

ADVANTAGE - Implementation is selected by automatically analyzing semantics of the intentional program tree. An accurate expression of the programmer's intent is achieved. The intentional program tree is created by direct manipulation.

DESCRIPTION OF DRAWING(S) - The figure represents a state diagram illustrating the navigation of terminal operands.

☐ 4. Document ID: US 5911072 A

L6: Entry 4 of 4 File: DWPI Jun 8, 1999

DERWENT-ACC-NO: 1999-393972
DERWENT-WEEK: 200366

TITLE: Program tree reduction method for high level programming

INVENTOR: SIMONYI, C

PRIORITY-DATA: 1995US-0431049 (April 28, 1995), 1993US-0145689 (October 29, 1993), 1997US-0884443 (June 27, 1997)

PATENT-FAMILY:

PUB-NO	PUB-DATE	LANGUAGE	PAGES	MAIN-IPC
US 5911072 A	June 8, 1999		056	G06F009/45

INT-CL (IPC): G06 F 9/45

ABSTRACTED-PUB-NO: US 5911072A
BASIC-ABSTRACT:

NOVELTY - Each node of subtree represents low level construct identified enzymes for corresponding constructs match functions which are invoked by passing a reference to node until function indicates that enzyme can reduced node.

DETAILED DESCRIPTION - A reduction enzyme is provided for each high level computational construct in a program tree for replacing a node representing construct with subtree. An INDEPENDENT CLAIM is also included for a recording medium in which program tree

reduction procedure is stored.

USE - For high level programming.

ADVANTAGE - (1) Semantics of computational constructs are conveyed correctly by display representation generator. Since program is stored in IP tree in syntax independent manner, it allows programmer to select from various programming languages.

DESCRIPTION OF DRAWING(S) - The figure shows block diagram of components of IP system.

Terms	Documents
L5	4

Hit List

Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

Search Results - Record(s) 1 through 20 of 21 returned.

☐ 1. Document ID: US 20040015929 A1

Using default format because multiple data bases are involved.

L10: Entry 1 of 21 File: PGPB Jan 22, 2004

PGPUB-DOCUMENT-NUMBER: 20040015929
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20040015929 A1

TITLE: Methods and systems for developing data flow programs

PUBLICATION-DATE: January 22, 2004

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY	RULE-47
Lewis, Brad R.	Broomfield	CO	US	
Boucher, Michael L.	Lafavette	CO	US	
Hinker, Paul	Longmont	CO	US	
Horton, Noah	Boulder	CO	US	

US-CL-CURRENT: 717/156; 717/159

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KWIC | Draw. Desc | Ir

☐ 2. Document ID: US 20020133497 A1

L10: Entry 2 of 21 File: PGPB Sep 19, 2002

PGPUB-DOCUMENT-NUMBER: 20020133497
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20020133497 A1

TITLE: Nested conditional relations (NCR) model and algebra

PUBLICATION-DATE: September 19, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY	RULE-47
Draper, Denise L.	Seattle	WA	US	
Horen, Gary	Seattle	WA	US	
Wagner, Tim	Seattle	WA	US	

US-CL-CURRENT: 707/100

ABSTRACT:

A method and system for providing data integration of multiple data stores with diverse formats. The data integration engine accepts queries using a standard query language such as XML-QL, executes those queries against the multiple data stores, and returns the results. The data stores may include relational databases, hierarchical databases, file systems, application data available via APIs, and so on. A query may reference data that resides in different data stores. The data integration engine allows operations such as joins across multiple data stores.

☐ 3. Document ID: US 20010037496 A1

L10: Entry 3 of 21 File: PGPB Nov 1, 2001

PGPUB-DOCUMENT-NUMBER: 20010037496
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20010037496 A1

TITLE: Method and system for generating a computer program

PUBLICATION-DATE: November 1, 2001

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY	RULE-47
Simonyi, Charles	Medina	WA	US	

US-CL-CURRENT: 717/146

ABSTRACT:

A method and system is described for generating executable code for a computer program. A programmer creates an intentional program tree using a syntax-independent editor. The editor allows a programmer to directly manipulate the intentional program tree. The intentional program tree has nodes. Each node represents a high-level computational construct of the computer program. For each node representing a high-level computational construct, the system transforms the node into an implementation of the high-level computational construct using low-level computational constructs. For each node representing a low-level computational construct, the system generates executable code that implements the low-level computational construct. The system further provides that where a high-level computational construct has a plurality of implementations of the high-level computational construct, the system transforms the nodes by selecting one of the implementations and transforms the node in accordance with the selected implementation. The system further provides that the implementation is selected by automatically analyzing semantics of the intentional program tree.

☐ 4. Document ID: US 6625804 B1

L10: Entry 4 of 21 File: USPT Sep 23, 2003

US-PAT-NO: 6625804
DOCUMENT-IDENTIFIER: US 6625804 B1

TITLE: Unified event programming model

DATE-ISSUED: September 23, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Ringseth; Paul F.	Redmond	WA		
Fernandez; Roland L.	Woodinville	WA		

US-CL-CURRENT: 717/114; 709/230, 717/104, 717/107, 717/140, 717/165, 719/310, 719/318

ABSTRACT:

A unified event programming model standardizes event programming for disparate eventing protocols. The unified event programming model simplifies programming events for different object types by abstracting away protocol-specific details. A protocol-independent compiler construct allows a programmer to specify events for an event source. Other protocol-independent compiler constructs allow a programmer to specify how to hook and unhook an event receiver from events. Based upon protocol-independent compiler constructs and an eventing protocol type value, a compiler generates an event source or event receiver implementation that is specific to an eventing protocol.

29 Claims, 25 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 23

☐ 5. Document ID: US 6618737 B2

L10: Entry 5 of 21 File: USPT Sep 9, 2003

US-PAT-NO: 6618737
DOCUMENT-IDENTIFIER: US 6618737 B2

TITLE: Speculative caching of individual fields in a distributed object system

DATE-ISSUED: September 9, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Aridor; Yariv	Haifa			IL
Factor; Michael	Haifa			IL
Eilam; Tamar	Haifa			IL
Schuster; Assaf	Haifa			IL

Teperman; Avi Haifa IL

US-CL-CURRENT: 707/205; 707/101

ABSTRACT:

This disclosure presents a technique of field-level caching in distributed object-oriented systems, in which a speculative approach is taken to identify opportunities for caching. The speculative approach is particularly suitable for exploitation of opportunities for caching. Invalidation protocols, which are fully compliant with the Java memory model, are provided to recover from incorrect speculation, while incurring only a low overhead. The technique has been implemented on a cluster of machines, and has been found to be readily scalable with multi-threaded applications. Field caching, optionally combined with other optimizations, produces a practically important performance step up in distributed environments, such as the cluster virtual machine for Java, which transparently distributes an application's threads and objects among the nodes of a cluster.

45 Claims, 16 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 9

☐ 6. Document ID: US 6615253 B1

L10: Entry 6 of 21 File: USPT Sep 2, 2003

US-PAT-NO: 6615253
DOCUMENT-IDENTIFIER: US 6615253 B1

TITLE: Efficient server side data retrieval for execution of client side applications

DATE-ISSUED: September 2, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Bowman-Amuah; Michel K.	Colorado Springs	CO		

US-CL-CURRENT: 709/219; 707/100, 711/118

ABSTRACT:

A system, method, and article of manufacture are provided for efficiently retrieving data. A total amount of data required for an application executed by a client is determined. In a single call, the total amount of data from a server is requested over a network. All of the data is bundled into a data structure by the server in response to the single call. The bundled data structure is sent to the client over the network and the data of the data structure is cached on the client. The cached data of the data structure is used as needed during execution of the application on the client.

18 Claims, 195 Drawing figures
Exemplary Claim Number: 1

Number of Drawing Sheets: 123

Full | Title | Citation | Front | Review | Classification | Date | Reference | Claims | KWIC | Draw Desc

☐ 7. Document ID: US 6601234 B1

L10: Entry 7 of 21 File: USPT Jul 29, 2003

US-PAT-NO: 6601234
DOCUMENT-IDENTIFIER: US 6601234 B1

TITLE: Attribute dictionary in a business logic services environment

DATE-ISSUED: July 29, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Bowman-Amuah; Michel K.	Colorado Springs	CO		

US-CL-CURRENT: 717/108; 705/7, 717/107, 717/116

ABSTRACT:

A system and method are provided for controlling access to data of a business object via an attribute dictionary. The attribute dictionary, which stores attribute names and values, is dispatched over a network. A helper facade is provided for interfacing a business object and the attribute dictionary. Next, it is verified that a current user is authorized to either set or get one of the attribute values upon a request which includes the attribute name that corresponds to the attribute value. The helper facade is called to set, get, or update one of the attribute values based on the corresponding attribute name, wherein the helper facade shields the attribute dictionary from the application code of the business object. The attribute value in the attribute dictionary is obtained or updated if the verification is successful, and a dirty flag is set in the attribute dictionary and an indicator is broadcast upon the attribute value being updated.

15 Claims, 195 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 123

Full | Title | Citation | Front | Review | Classification | Date | Reference | Claims | KWIC | Draw Desc

☐ 8. Document ID: US 6594783 B1

L10: Entry 8 of 21 File: USPT Jul 15, 2003

US-PAT-NO: 6594783
DOCUMENT-IDENTIFIER: US 6594783 B1

TITLE: Code verification by tree reconstruction

DATE-ISSUED: July 15, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Dollin; Christopher J	Yate Bristol			GB
Knight Leach; Steven Frederick	North Somerset			GB
Oberhauser; Roy T.	Santa Clara	CA		
Dickey; Laura J.	Westford	MA		

US-CL-CURRENT: 714/38; 717/156

ABSTRACT:

A system and method are provided that allow for improved code sequence verification through the use of an abstract syntax tree. This is accomplished by first constructing an abstract syntax tree from the code sequence and then determining whether the abstract syntax tree satisfies a predefined set of conditions indicative of the code sequence being executable on the computer without generating a predefined class of execution errors. The abstract syntax tree is constructed by reassembling the code sequence into a plurality of instructions, combining the instructions into a plurality of blocks, examining the blocks to determine entry points of a plurality of loops, and tagging locations in the series of instructions where control is transferred at the end of each loop. The instructions, blocks, loops and tagged locations are then examined to generate a plurality of control structures (the coarse structure). Finally, the instructions, blocks, loops, tagged locations and control structures are examined to generate a plurality of form expressions (the fine structures).

23 Claims, 33 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 31

☐ 9. Document ID: US 6539396 B1

L10: Entry 9 of 21 File: USPT Mar 25, 2003

US-PAT-NO: 6539396
DOCUMENT-IDENTIFIER: US 6539396 B1

TITLE: Multi-object identifier system and method for information service pattern environment

DATE-ISSUED: March 25, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Bowman-Amuah; Michel K.	Colorado Springs	CO		

US-CL-CURRENT: 707/103R; 707/101, 707/102, 707/103X, 707/202, 707/203, 711/118

ABSTRACT:

A system and method for implementing an association of business objects without retrieving said objects from a database on which they are stored. A business object in the business cache is provided and an instance of an associated object is stored on a database. An association of the business object with the instance of the associated object is determined. An object identifier is generated containing information including the determination association which is necessary to retrieve the instance of the associated object from the database, wherein the object identifier includes a unique row identifier, an identifier generated by a utility, and a unique string generated from one or more attributes. The object identifier is loaded when the business object starts. A location of the instance of the associated object on the database is determined from the object identifier and the instance of the associated object is retrieved from the database.

15 Claims, 195 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 123

☐ 10. Document ID: US 6496850 B1

L10: Entry 10 of 21 File: USPT Dec 17, 2002

US-PAT-NO: 6496850
DOCUMENT-IDENTIFIER: US 6496850 B1

TITLE: Clean-up of orphaned server contexts

DATE-ISSUED: December 17, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Bowman-Amuah; Michel K.	Colorado Springs	CO		

US-CL-CURRENT: 709/203; 707/102, 709/224, 709/228

ABSTRACT:

A system, method and article of manufacture are provided for detecting an orphaned server context. A collection of outstanding server objects is maintained and a list of contexts is created for each of the outstanding server objects. A compilation of clients who are interested in each of the outstanding server objects are added to the list. Recorded on the list is a duration of time since the clients invoked a method accessing each of the contexts of the outstanding server objects. The list is examined at predetermined intervals for determining whether a predetermined amount of time has passed since each of the objects has been accessed. Contexts that have not been accessed in the predetermined amount of time are selected and information is sent to the clients identifying the contexts that have not been accessed in the predetermined amount of time.

19 Claims, 195 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 123

☐ 11. Document ID: US 6346945 B1

L10: Entry 11 of 21 File: USPT Feb 12, 2002

US-PAT-NO: 6346945
DOCUMENT-IDENTIFIER: US 6346945 B1
**** See image for Certificate of Correction ****

TITLE: Method and apparatus for pattern-based flowcharting of source code

DATE-ISSUED: February 12, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Mansurov; Nikolai	Moscow			RU
Campara; Djenana	Nepean			CA
Rajala; Norman	Nepean			CA

US-CL-CURRENT: 345/473; 717/146

ABSTRACT:

A system and method for generating a consistent graphical expression of source code which is independent of the source language and of a particular programmer's style. The system first provides an intermediary pattern language which is source language independent into which the source code is translated. This pattern language is directly mapped to a set of predetermined graphical patterns having a series of attributes. The pattern language is nested in the sense that certain expressions may contain certain other expressions. This translates directly to graphical containment. Attributes are computed starting with the most nested parts of the pattern language translation. The attributes of the more nested parts having been thus computed may be used in the computation of the attributes of less nested parts of the expression. Once all attributes are computed, a mapping to display directives is performed, and a graphical engine produces an actual display.

8 Claims, 18 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 18

Full | Title | Citation | Front | Review | Classification | Date | Reference | Claims | KWIC | Draw. Desc | Ir

☐ 12. Document ID: US 6314562 B1

L10: Entry 12 of 21 File: USPT Nov 6, 2001

US-PAT-NO: 6314562
DOCUMENT-IDENTIFIER: US 6314562 B1
**** See image for Certificate of Correction ****

TITLE: Method and system for anticipatory optimization of computer programs

DATE-ISSUED: November 6, 2001

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Biggerstaff; Ted J.	Bellevue	WA		

US-CL-CURRENT: 717/156; 717/144, 717/151

ABSTRACT:

A method and system for anticipatory optimization of computer programs. The system generates code for a program that is specified using programming-language-defined computational constructs and user-defined, domain-specific computational constructs. The system generates an abstract syntax tree (AST) representation of the program. The AST has nodes representing the computational constructs of the program. For each user-defined, domain-specific computational construct, the system determines whether a user-defined, domain-specific transform has been defined for the computational construct. The transform transforms a portion of the AST relating to the user-defined, domain-specific computational construct into one or more programming-language-defined computational constructs. When a domain-specific transform has been defined for the computational construct, the system transforms the AST in accordance with the domain-specific transform. The transformed AST is in a form that reflects an optimization of the programming-language-defined computational constructs based on the user-defined, domain-specific computational construct.

53 Claims, 50 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 49

☐ 13. Document ID: US 6189143 B1

L10: Entry 13 of 21 File: USPT Feb 13, 2001

US-PAT-NO: 6189143
DOCUMENT-IDENTIFIER: US 6189143 B1

TITLE: Method and system for reducing an intentional program tree represented by high-level computational constructs

DATE-ISSUED: February 13, 2001

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Simonyi; Charles	Medina	WA		

US-CL-CURRENT: 717/109; 717/144

ABSTRACT:

A method and system is described for generating executable code for a computer program, A programmer creates an intentional program tree using a syntax-independent editor. The editor allows a programmer to directly manipulate the intentional program tree. The intentional program tree has nodes. Each node represents a high-level computational construct of the computer program. For each node representing a high-level computational construct, the system transforms the node into an implementation of the high-level computational construct using low-level computational constructs. For each node representing a low-level computational construct, the system generates executable code that implements the low-level computational construct. The system further provides that where a high-level computational construct has a plurality of implementations of the high-level computational construct, the system transforms the nodes by selecting one of the implementations and transforms the node in accordance with the selected implementation. The system further provides that the implementation is selected by automatically analyzing semantics of the intentional program tree.

64 Claims, 38 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 35

☐ 14. Document ID: US 6097888 A

L10: Entry 14 of 21 File: USPT Aug 1, 2000

US-PAT-NO: 6097888
DOCUMENT-IDENTIFIER: US 6097888 A

TITLE: Method and system for reducing an intentional program tree represented by high-level computational constructs

DATE-ISSUED: August 1, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Simonyi; Charles	Medina	WA		

US-CL-CURRENT: 717/144; 345/967, 717/146

ABSTRACT:

A method and system for generating a computer program in the manner that uses no computer programming language syntax. The system represents a computer program as an intentional program tree, which is a high-level program tree that is a syntax-independent representation using high-level computational constructs. The intentional program tree represents a programmer's intent, rather than an implementation of the programmer's intent. The programmer creates an intentional program tree using a syntax-independent editor. The editors allows a programmer to directly

manipulate the intentional program tree. Because the program is stored as an intentional program tree in a syntax-independent manner, the editor allows the program to select in which of a various programming language the computer program is to be displayed. In addition, the system transforms an intentional program tree to a reduced program tree, which is a program tree comprising low-level computational constructs, in a process

called reduction. The reduction process replaces expressions of programmer's intents with a representation of one of possible multiple implementations of those intents using low-level computational constructs.

55 Claims, 38 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 35

☐ 15. Document ID: US 6078746 A

L10: Entry 15 of 21 File: USPT Jun 20, 2000

US-PAT-NO: 6078746
DOCUMENT-IDENTIFIER: US 6078746 A

TITLE: Method and system for reducing an intentional program tree represented by high-level computational constructs

DATE-ISSUED: June 20, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Simonyi; Charles	Medina	WA		

US-CL-CURRENT: 717/144; 717/114

ABSTRACT:

A method and system for generating a computer program in the manner that uses no computer programming language syntax. The system represents a computer program as an intentional program tree, which is a high-level program tree that is a syntax-independent representation using high-level computational constructs. The intentional program tree represents a programmer's intent, rather than an implementation of the programmer's intent. The programmer creates an intentional program tree using a syntax-independent editor. The editors allows a programmer to directly manipulate the intentional program tree. Because the program is stored as an intentional program tree in a syntax-independent manner, the editor allows the program to select in which of a various programming language the computer program is to be displayed. In addition, the system transforms an intentional program tree to a reduced program tree, which is a program tree comprising low-level computational constructs, in a process called reduction. The reduction process replaces expressions of programmer's intents with a representation of one of possible multiple implementations of those intents using low-level computational constructs.

16 Claims, 38 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 35

☐ 16. Document ID: US 6029002 A

L10: Entry 16 of 21 File: USPT Feb 22, 2000

US-PAT-NO: 6029002
DOCUMENT-IDENTIFIER: US 6029002 A

TITLE: Method and apparatus for analyzing computer code using weakest precondition

DATE-ISSUED: February 22, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Afifi; Ashraf	Burlington	MA		
Chan; Dominic	Carlisle	MA		
Comuzzi; Joseph J.	Groton	MA		
Hart; Johnson M.	Weston	MA		
Pizzarello; Antonio	Phoenix	AZ		

US-CL-CURRENT: 717/131; 717/125, 717/144, 717/146

ABSTRACT:

An analyzer for maintaining and analyzing source code is disclosed. The analyzer includes a software translator for converting conventional source code into an intermediate language, slicing capability based upon weakest precondition determination, dual direction flow analysis and incorporation of a computational model to facilitate iterative code.

22 Claims, 95 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 63

☐ 17. Document ID: US 5842204 A

L10: Entry 17 of 21 File: USPT Nov 24, 1998

US-PAT-NO: 5842204
DOCUMENT-IDENTIFIER: US 5842204 A

TITLE: Method and apparatus for translating source code from one high-level computer language to another

DATE-ISSUED: November 24, 1998

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Andrews; Kristy A.	Palo Alto	CA		

Del Vigna; Paul	San Jose	CA
Molloy; Mark E.	San Jose	CA

US-CL-CURRENT: 707/3; 341/51, 341/79, 707/102

ABSTRACT:

A method and apparatus for translating source code written in one computer language to source code written in another language wherein translated static fragments are generated in the face of textual inconsistencies. Exactly one target language definition of each source language static fragment is generated and the differences are encapsulated in new parameters.

21 Claims, 22 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 12

☐ 18. Document ID: US 5822593 A

L10: Entry 18 of 21 File: USPT Oct 13, 1998

US-PAT-NO: 5822593
DOCUMENT-IDENTIFIER: US 5822593 A

TITLE: High-level loop fusion

DATE-ISSUED: October 13, 1998

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Lamping; John O.	Los Altos	CA		
Kiczales; Gregor J.	Palo Alto	CA		
Mendhekar; Anurag D.	Sunnyvale	CA		

US-CL-CURRENT: 717/161; 717/144, 717/156

ABSTRACT:

A processor is provided with a software program specifying an overall computation that includes operations. Each operation implies a set of subcomputations, without explicitly specifying a control structure for carrying out the subcomputations according to a particular sequencing. The operations include a first and a second operation, and the provided software program further specifies how the first and second operations are combined in the overall computation. For example, the first and second operations can each imply, respectively, a first and a second computational loop, the first loop including the subcomputations of the first operation, the second loop including the subcomputations of the second operation. A description of possible sequencings of subcomputations of the first and second operations is provided, to be used in implementing the specified combination of the first and second operations, the description including a set of constraints on the sequencing of subcomputations of the first and second operations. A software program is automatically generated that includes

a combined operation implementing the specified combination of the first and second operations. The combined operation has a control structure for carrying out the subcomputations of the first and second operations in accordance with the constraints. This control structure can be, for example, a computational loop. If the first and second operations imply, respectively, first and second computational loops, the control structure of the combined operation can be, for example, a computational loop including a fusion of the first and second loops.

28 Claims, 11 Drawing figures
Exemplary Claim Number: 28
Number of Drawing Sheets: 9

☐ 19. Document ID: US 5790863 A

L10: Entry 19 of 21 File: USPT Aug 4, 1998

US-PAT-NO: 5790863
DOCUMENT-IDENTIFIER: US 5790863 A

TITLE: Method and system for generating and displaying a computer program

DATE-ISSUED: August 4, 1998

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Simonyi; Charles	Medina	WA		

US-CL-CURRENT: 717/113; 700/83, 700/87, 717/114, 717/156

ABSTRACT:

A method and system for generating a computer program. In a preferred embodiment, the present invention provides a program tree editor for directly manipulating a program tree. A program tree comprises of plurality of nodes corresponding to computational constructs. The program tree editor receives commands from a user that are independent of a programming language syntax. The present invention also provides a display representation generator for generating a display representation of the program tree. The display representation generator retrieves nodes from the program tree and displays a display representation of the node. A user of the present invention preferably interacts with the program tree editor based on the display representation.

15 Claims, 20 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 18

☐ 20. Document ID: US 5428793 A

US-PAT-NO: 5428793
DOCUMENT-IDENTIFIER: US 5428793 A

TITLE: Method and apparatus for compiling computer programs with interproceduural register allocation

DATE-ISSUED: June 27, 1995

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Odnert; Daryl	Boulder Creek	CA		
Santhanam; Vatsa	Sunnyvale	CA		

US-CL-CURRENT: 717/157

ABSTRACT:

Optimization techniques are implemented by means of a program analyzer used in connection with a program compiler to optimize usage of limited register resources in a computer processor. The first optimization technique, called interprocedural global variable promotion allows the global variables of a program to be accessed in common registers across a plurality of procedures. Moreover, a single common register can be used for different global variables in distinct regions of a program call graph. This is realized by identifying subgraphs, of the program call graph, called webs, where the variable is used. The second optimization technique, called spill code motion, involves the identification of regions of the call graph, called clusters, that facilitate the movement of spill instructions to procedures which are executed relatively less often. This decreases the overhead of register saves and restores which must be executed for procedure calls.

4 Claims, 9 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 7

Search Results

Search Results for: **[(intent% program%)]**
Found **5** of **127,132 searched.**

Search within Results

1 An issue-oriented approach to judicial document assembly 87%
L. Karl Branting
Proceedings of the fourth international conference on Artificial intelligence and law
August 1993

2 Focus: non-photorealistic rendering: Intentional non-photorealistic rendering 80%
Dan Goldstein
ACM SIGGRAPH Computer Graphics February 1999
Volume 33 Issue 1
Non-photorealistic rendering (NPR) is a subject that has been generating quite a bit of interest in the computer graphics community lately. This is partly because any advance in the area is so rapidly picked up by authors who create actual content. The term NPR covers any computer-generated imagery explicitly rendered using techniques designed not to mimic physical reality. Most frequently, NPR rendering is performed by a program that takes images or three-dimensional geometry as input and creat ...

3 The runtime creation of code for printing simulation output 77%
John H. Reynolds
Proceedings of the 22nd conference on Winter simulation December 1990

4 Annotating objects for transport to other worlds 77%
David Ungar
ACM SIGPLAN Notices , Proceedings of the tenth annual conference on Object-oriented programming systems, languages, and applications October 1995
Volume 30 Issue 10
In Self 4.0, people write programs by directly constructing webs of objects in a larger world of objects. But in order to save or share these programs, the objects must be moved to other worlds. However, a concrete, directly constructed program is incomplete, in particular missing five items of information: which module to use, whether to transport an actual value or a counterfactual initial value, whether to create a new object in the new world or to refer to an existing one, whether an object ...

5 Supporting interactivity in automated 3D illustrations 77%

Dorée Duncan Seligmann , Steven Feiner

Proceedings of the 1st international conference on Intelligent user interfaces February 1993

Results 1 - 5 of 5 **short listing**

Your search matched **3** of **1003743** documents.
A maximum of **500** results are displayed, **15** to a page, sorted by **Relevance** in **Descending** order.

Refine This Search:
You may refine your search by editing the current search expression or entering a new one in the text box.

intentional programming Search

☐ Check to search within this result set

Results Key:
JNL = Journal or Magazine **CNF** = Conference **STD** = Standard

---

1 **Transformation in intentional programming**
Aitken, W.; Dickens, B.; Kwiatkowski, P.; De Moor, O.; Richter, D.; Simonyi, C.;
Software Reuse, 1998. Proceedings. Fifth International Conference on , 2-5 June 1998
Pages:114 - 123

[Abstract] [PDF Full-Text (1664 KB)] **IEEE CNF**

---

2 **Understanding users intention: programming fine manipulation tasks by demonstration**
Zollner, R.; Rogalla, O.; Dillmann, R.; Zollner, M.;
Intelligent Robots and System, 2002. IEEE/RSJ International Conference on , Volume: 2 , 30 Sept.-5 Oct. 2002
Pages:1114 - 1119 vol.2

[Abstract] [PDF Full-Text (788 KB)] **IEEE CNF**

---

3 **I&C upgrade measures for VVER model V213 units in the Czech Republic, Hungary and Slovakia**
Parsons, T.;
Eastern European Nuclear Power Plant Standards., IEE Seminar on (Digest No: 1996/046) , 14 Feb. 1996
Pages:5/1 - 5/5

[Abstract] [PDF Full-Text (272 KB)] **IEE CNF**

Searched pages from **citeseer.nj.nec.com** for **intentional programming**. Results **1 - 10** of about **890**. Search took **0.3**

The Death Of Computer Languages, The Birth Of **Intentional** ...
... reasons. First, currently I am only articulate when I discuss **Intentional Programming** so this limited the choice of subjects. But ...
citeseer.nj.nec.com/simonyi95death.html - 22k - Cached - Similar pages

Citations: the birth of **intentional programming** - Simonyi, of ...
C. Simonyi. The death of computer languages, the birth of **intentional programming**. ... The
death of **programming** languages, the birth of **intentional programming**. ...
citeseer.nj.nec.com/context/132413/0 - 33k - Cached - Similar pages

Citations: Transformation in **Intentional Programming** - Aitken ...
W. Aitken, B. Dickens, P. Kwiatkowski, O. de Moor, D. Richter, and C. Simonyi, Transformation in **Intentional Programming**, 5th Int. Conf. Softw. ...
citeseer.nj.nec.com/context/322416/0 - 24k - Cached - Similar pages

Intentional Programming - Innovation in the Legacy Age - Simonyi ...
... (a systems of concerns for objects has been defined) type parameterization (static) and delegation (dynamic) **Intentional Programming** [Sim96], Sim95] static ...
citeseer.nj.nec.com/simonyi96intentional.html - 17k - Cached - Similar pages

Intentional Programming: a Host of Language Features - Van Wyk, de ...
This paper This paper is a report to our colleagues in the **Intentional Programming** team at Microsoft about a model and prototype implementation of **intentional** ...
citeseer.nj.nec.com/vanwyk01intentional.html - 22k - Cached - Similar pages

Citations: The death of computer languages, the birth of ...
C. Simonyi, The death of computer languages, the birth of **Intentional Programming** , Tech. Rep. MSR-TR-95-52, Microsoft Research, 1995. 10 citations found. ...
citeseer.nj.nec.com/context/506638/52171 - 20k - Cached - Similar pages

Beyond Objects: Generative **Programming** - Czarnecki, Eisenecker ...
... 1995 24 Validating Component Compositions in Software System Generat.. - Batory, Geraci 21 The Birth of **Intentional Programming** (context) - Simonyi, of ...
citeseer.nj.nec.com/408302.html - 23k - Cached - Similar pages

Forwarding in Attribute Grammars for Modular Language Design - Van ...
... are traditionally... Cited by: More **Intentional Programming**: a Host of Language Features - Van Wyk, de Moor.. (2001) (Correct) Active ...
citeseer.nj.nec.com/vanwyk02forwarding.html - 23k - Cached - Similar pages

Citations: Department of Information and Computer Science ...
... 5.2 **Intentional Programming Intentional programming** is .... JM Neighbors. ... 5.2 **Intentional Programming Intentional programming** .... JM Neighbors. ...
citeseer.nj.nec.com/context/322425/0 - 15k - Cached - Similar pages

Specification Languages in Algebraic Compilers (ResearchIndex)
... Language.. - Van Wyk, de Moor.. (2002) (Correct) 0.8: **Intentional Programming**:

a Host of Language Features - Van Wyk, de Moor.. (2001 ...
citeseer.nj.nec.com/vanwyk00specification.html - 24k - Cached - Similar pages

Goooooooooogle ▶
Result Page: 1 2 3 4 5 6 7 8 9 10 **Next**

site:citeseer.nj.nec.com intentional p | Google Search | Search within results

Dissatisfied with your search results? Help us improve.

Get the Google Toolbar: 

Google Home - Advertise with Us - Business Solutions - Services & Tools - Jobs, Press, & Help